Parallel population-based algorithm portfolios:

Akay, Rustu; Basturk, Alper; Kalinli, Adem; Yao, Xin

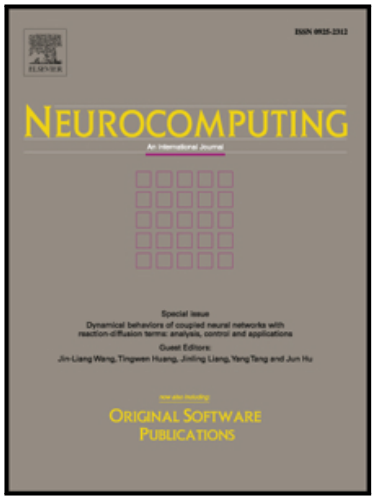
DOI:
10.1016/j.neucom.2017.03.061

Document Version
Peer reviewed version

Citation for published version (Harvard):
Akay, R, Basturk, A, Kalinli, A & Yao, X 2017, 'Parallel population-based algorithm portfolios: An empirical study', *Neurocomputing*, vol. 247, pp. 115-125. https://doi.org/10.1016/j.neucom.2017.03.061

Communicated by Dr Ding Wang

Accepted Manuscript



Parallel population-based algorithm portfolios: An empirical study

Rustu Akay, Alper Basturk, Adem Kalinli, Xin Yao

PII: S0925-2312(17)30594-5
DOI: 10.1016/j.neucom.2017.03.061
Reference: NEUCOM 18294

To appear in: *Neurocomputing*

Received date: 8 February 2016
Revised date: 31 January 2017
Accepted date: 23 March 2017

Please cite this article as: Rustu Akay, Alper Basturk, Adem Kalinli, Xin Yao, Parallel population-based algorithm portfolios: An empirical study, *Neurocomputing* (2017), doi: 10.1016/j.neucom.2017.03.061

Parallel population-based algorithm portfolios: An empirical study

Rustu Akay[a,*], Alper Basturk[b], Adem Kalinli[b], Xin Yao[c]

[a]*Department of Mechatronics Engineering, Erciyes University, Kayseri, Turkey*
[b]*Department of Computer Engineering, Erciyes University, Kayseri, Turkey*
[c]*School of Computer Science, University of Birmingham, Birmingham, B15 2TT, U.K.*

Abstract

Although many algorithms have been proposed, no single algorithm is better than others on all types of problems. Therefore, the search characteristics of different algorithms that show complementary behavior can be combined through portfolio structures to improve the performance on a wider set of problems. In this work, a portfolio of the Artificial Bee Colony, Differential Evolution and Particle Swarm Optimization algorithms was constructed and the first parallel implementation of the population-based algorithm portfolio was carried out by means of a Message Passing Interface environment. The parallel implementation of an algorithm or a portfolio can be performed by different models such as master-slave, coarse-grained or a hybrid of both, as used in this study. Hence, the efficiency and running time of various parallel implementations with different parameter values and combinations were investigated on benchmark problems. The performance of the parallel portfolio was compared to those of the single constituent algorithms. The results showed that the proposed models reduced the running time and the portfolio delivered a robust performance compared to each constituent algorithm. It is observed that the speedup gained over the sequential counterpart changed significantly depending on the structure of the portfolio. The portfolio is also applied to a training of neural networks which has been used for time series prediction. Result demonstrate that, portfolio is

*Corresponding author : akay@erciyes.edu.tr (Rustu Akay)

able to produce good prediction accuracy.

Keywords: Algorithm portfolios, global optimization, neural networks, parallel computing.

1. Introduction

In the last few decades, different kinds of EAs have been developed. These algorithms have shown excellent search abilities but usually their performance may vary considerably according to the problem. No single algorithm performs better than others. Therefore, selection of the most appropriate algorithm for a particular problem needs considerable time. In this case, to solve problems efficiently, researchers use a combination of different algorithms, which is usually referred to as Portfolio Algorithm (PA) [1, 2, 3, 4]. The main aim of these portfolio algorithms is to improve the performance and to make a more stable algorithm by combining the advantages of the optimization algorithms. Although these portfolios provide better results than the single algorithms of the constituting portfolios, their main problem is that their running times to solve large scale problems are unacceptable. Parallel computing systems, which emerged as a result of recent hardware developments, can be employed to overcome this problem. Parallel computing divides tasks into smaller parts and operates the parts on multiprocessor hardware architectures simultaneously. In this way, increased computing needs are met and the solutions are obtained faster. On the other hand, parallel programming has grown even more in popularity because of the availability of multi-core CPUs and it is known that EAs can explore efficiently the use of parallel concepts to speed the process. The main purpose of parallel EAs is to solve a given problem in less time.

Although the portfolio idea is not new, there has not been much research done in parallel implementation of portfolios. A parallel portfolio of algorithms is a collection of different algorithms running on different CPUs. The main aim of the parallel portfolio is not only to enhance the performance of the component algorithms but also to reduce the computation time.

In this study, a Parallel Portfolio Algorithm (PPA) was proposed on different parallelization models through a Message Passing Interface (MPI) [5] among multiprocessing units. We study the possibility of combining algorithms into portfolios and implementing a parallel portfolio including the Artificial Bee Colony (ABC) [6], Differential Evolution (DE) [7] and Particle Swarm Optimization (PSO) [8], which are recent evolutionary computation techniques and of which the speedup performance are analyzed with these parallelization models. Many different versions of these algorithms based on different models have been proposed and used to solve different optimization problems in the literature. Some survey papers have summarized these studies [9, 10, 11, 12] and some papers carried out a review on main metaheuristics and presented their similarities and differences [13].

As mentioned before, no one algorithm outperforms all other algorithms on all types of problems. Some problems that are unsolvable by superior algorithm can be solved with an inferior one. This fact has encouraged researchers to work on portfolio algorithm structures. The earliest portfolio algorithm approach was proposed by Gomes and Selman [1] and Huberman et al. [2] in 1997. These studies applied a portfolio of two or more Las Vegas algorithms to certain combinatorial problems and obtained significant performance improvement. Fukunaga proposed combining different genetic algorithm instances with different control parameter sets and applied them to solve the traveling salesmen problem [3]. Subsequent work focused on showing the computational advantage of a portfolio approach on hard combinatorial search problems [4]. In the following years, a portfolio algorithm for numerical optimization was proposed by Peng et al. [14]. This study emphasized that, selecting the best algorithm is very difficult in a limited budget time and that distributing the time budget to multiple algorithms can reduce this risk. Vrugt and Robinson proposed an adaptive method approach that combine genetic algorithm, evolution strategy and particle swarm optimization in one framework [15]. Some studies that combine a variety of genetic algorithms use population partitioning techniques to obtain better performance [16, 17]. Some studies focused on integrating multiple search

operators or algorithms in a portfolio [18, 19, 20]. Samulowitz et al. developed an algorithm portfolio tool called snappy that is provide a strong baseline and easily extended by its users [21]. More recently, Tang et al. [22] proposed algorithm portfolios based on the Estimated Performance Matrix (EPM-PAP), which is equipped with a novel constituent algorithms selection module. In this paper the proposed model can successfully identify the appropriate constituent algorithm and achieve better results than all the single algorithms considered in their study. Not only selection of the most appropriate algorithm but also their associated control parameter settings for a particular problem needs considerable time. Extensive studies have been done on appropriate setting of the control parameters of EAs [23, 24, 25, 26, 27]. Also some studies focused on adaptive strategies for control parameter selection [28, 29]. Also portfolio algorithm is related to ensemble methods that use multiple learning algorithms to obtain better predictive performance [30, 31]. In ensemble methods while some studies combine different algorithm like algorithm portfolio [32, 33], the others combine same algorithms with different parameters or adaptive strategies [34, 35, 36]. Zhao et al. proposed an ensemble of different neighborhood sizes with online self-adaptation to enhance the multiobjective evolutionary algorithm based on decomposition [37]. Shang et al. proposed a multi-population based cooperative coevolutionary algorithm to solve the multi-objective capacitated arc routing problem [38]. Portfolio algorithm is both effective and robust, but their parallel implementation has received relatively little attention. Petrik and Zilberstein [39] proposed a method for finding a static parallel portfolio of algorithms, in which the share of processor time allocated to each algorithm is fixed. Yun and Epstein [40] proposed an approach that constructs algorithm portfolios for parallel runs based on a combination of a greedy algorithm, and three heuristics. Also portfolio algorithm has been applied to solve various optimization problems from various fields, such as satisfiability problem [41], classification and prediction [42, 43, 44], and scheduling problem [45].

As a result, portfolio algorithms and ensemble methods naturally attract increasing attention. The application of the techniques in previous literature

is aimed to improve the performance of EAs while our study is aimed to not only on improving the performance of the constituent algorithms but also on investigating the PPAs running time in a parallel implementation context. For this purpose, we performed different parallelization models such as master-slave and coarse-grained and made a comparison among the serial and parallel PPAs. The other purpose of the study is to employ the proposed model to a real-world problem. The real-word problem, training a artificial neural network for predicting Mackey Glass and Box Jenkins time series. Althought many algorithms have been applied successfully for prediction problems [46, 47, 48, 49], parallel portfolio algorithms is not available in the literature yet.

The rest of the paper is organized as follows. In Section II, the EAs which are used in the proposed model are described briefly. Section III provides a description of the parallel implementation concept and the proposed parallel implementation portfolio is explained. The experiments and results are presented in Section IV. Section V contains some conclusions.

2. Algorithms

Some recent EAs include ABC, DE and PSO. A brief description of these algorithms is presented in the following paragraphs.

2.1. Artificial bee colony

The ABC algorithm was introduced by Karaboga in 2005 and is inspired by the foraging behavior of honey bees [6]. The working process of the ABC algorithm contains three different phases. The first phase is that of the employed bees which are responsible for exploiting all the sources. The second phase is that of the onlooker bees which use potentially rich sources. The third phase is that of the scout bees which are responsible for exploring undiscovered sources. "Limit", which is a parameter of the ABC algorithm, uses exhausted sources. In the ABC algorithm, employed bees and onlooker bees use the same perturbation strategy which is defined by Eq. (1).

$$x_{ij}(t+1) = x_{ij}(t) + rand()(x_{ij}(t) - x_{r_1 j}(t)) \quad (1)$$

where r_1 is the randomly selected solution vector ($r_1 \neq i$) and j is the randomly selected index. The rand() function generates a uniformly distributed random variable $\in [0, 1]$. Detailed information on the ABC algorithm can be found in [6, 50].

2.2. Differential evolution

The DE algorithm was introduced by Storn and Price in 2005 [7]. It uses different recombination operators to improve the population. In this paper we use the DE/rand/1/bin configuration. This configuration's update formulas are given by Eqs. (2) and (3).

$$v_i(t) = x_{r_1}(t) + F(x_{r_2}(t) - x_{r_3}(t)) \quad (2)$$

$$x_i(t+1) = \begin{cases} v_i(t) & \text{if } rand() \leq CR \\ x_i(t) & \text{else} \end{cases} \quad (3)$$

The weighting factor $F \in [0, 2]$ controls the amplification of differential variation. The crossover rate $CR \in [0, 1]$ probabilistically controls the amount of recombination. x_{r_1}, x_{r_2} and x_{r_3} are randomly selected solution vectors ($r_1 \neq r_2 \neq r_3 \neq i$). The rand() function generates a uniformly distributed random variable $\in [0, 1]$. Detailed information on the DE algorithm can be found in [7].

2.3. Particle swarm optimization

The PSO algorithm, which was introduced by Kennedy et al. in 1995, inspired by social and cooperative behavior displayed by various species like birds, fish etc [8]. It modifies velocities based on personal best and global best. The modification of velocities and positions are obtained by using Eqs. (4) and (5).

$$x_{id} = x_{id} + v_{id} \quad (4)$$

$$v_{id} = wv_{id} + c_1 r_1 (x_i^{pb} - x_{id}) + c_2 r_2 (x^{gb} - x_{id}) \quad (5)$$

where x_{id} is the i^{th} particle's position and v_{id} is the velocity for the i^{th} particle. x^{pb} and x^{gb} are the personal best and global best positions respectively. w is the inertia weight, which determines how the previous velocity of the particle influences the velocity in the next iteration. c_1 and c_2 are acceleration constants. r_1 and r_2 are the uniformly generated random number in the range of $\in [0, 1]$. Detailed information on the PSO algorithm can be found in [8].

3. Parallel models of portfolio algorithms

Parallel EAs can be performed based on different models which are categorized depending on the synchronization and the communication behaviors of subtasks. In this section, we briefly introduce parallel EA models and discuss the differences between them. In general, there are tree different well-known models for the parallelization of EAs which was named master-slave, fine-grained and coarse-grained [51]. These models are shown in Figure 1.

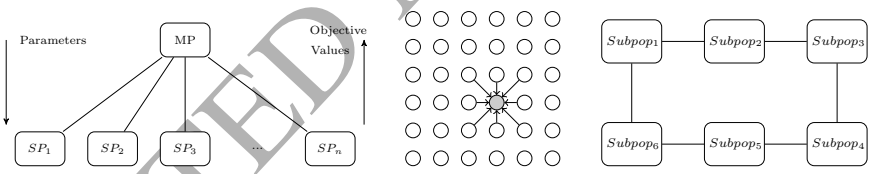


Figure 1: Parallel computing models, a)Master-slave, b)Fine-grained, and c)Coarse-Grained (MP:Master processor, SP:Slave Processor)

In the master-slave parallelization approach, while the evaluation process of each individual in the population is performed on a different slave processor, the selection and recombination processes take place in the master processor as given in Figure 1(a). The fitness value of each individual is independent of the others and there is no communication during the evaluation of the objective function. After evaluation, the slave processor sends its results to the master processor and the master processor performs the selection and recombination operations

to create offsprings. The offsprings are distributed to slave processors to be assigned a fitness value, and the process continues in this way. Communication occurs only when each slave sends the fitness value and receives its subset of individuals to evaluate.

In the fine-grained parallelization model, the population is distributed over the processors. Parallelism and communication are realized among some individuals a local neighborhood structure as given in Figure 1(b). The improvement when more processors used is limited. Using more processors for a small problem increases the running time significantly as a result of excessive communication between processors. That is, an individual can only compete and mate with its nearby neighbors. The topology of the neighborhood and the number of individuals in the neighborhood affects the performance of the parallel algorithm. This model is initially designed for working in massively parallel machines.

In the coarse-grained parallelization model the algorithms are executed concurrently on several independent subpopulations and they regularly exchange some individuals with those from other subpopulations during their search as given in Figure 1(c). In this model, new control parameters such as the number of individuals to migrate, the migration interval that controls the migration frequency and the topology which defines the neighbors of each island need to be set carefully.

Many hybrids have been defined by combining the above models, such as coarse-grained and fine-grained, coarse-grained and master-slave, and coarse-grained at two levels. The hybrid approach allows the advantages of two models to be combined.

All parallelization models have advantages and disadvantages. Parallel implementation of EAs may be complex and there are some problems with regard to selecting the parallel computing model and its optimum parameter setting.

In the master-slave model, the algorithms' search behavior does not change, it is exactly like a serial. As with other models, this model does not require additional parameters. This model is most suitable whenever the fitness evaluations are significantly expensive. The communication overhead between the master

processor and the slave processors becomes significant. The fine-grained model is designed for working in massively parallel machines and it can easily parallelize many features of EAs such as selection, mating, survival etc. The speedup and performance of the algorithms may significantly change the models' parameters: topology, neighborhood size and complicated shape. The coarse-grained model is easy to implement. When the fitness function is computationally inexpensive, this model is probably the most preferred. It needs less communication between nodes, so its efficiency may be better than those of other models. However, when EAs are parallelized based on the coarse-grained model, more control parameters are introduced to the approach, as mentioned before. The speedup and performance of the algorithms based on the coarse-grained model may significantly change due to the differences in the algorithms' behaviors. Therefore, additional control parameters need to be set carefully. Hybrid models generally to achieve the combined advantages of two models and give better performance than any of them alone.

3.1. Parallel population-based algorithm portfolios

A parallel portfolio of algorithms is a collection of different algorithms running on parallel computing systems. Their parallel implementation can be performed by different models. In this study, we explored two types of these models: coarse-grained and a hybrid model.

In our first model, the coarse-grained parallelization model was used to implement the parallel portfolio algorithm, in which each node runs the different algorithms independently and is initialized by its own set. Migration of individuals among sub-algorithms is performed in a certain interval to encourage the exchange experience of each independent algorithm among the sub-algorithms. The migration scheme adopted in the study uses a global sorting step in which the worst individual in mth sub-algorithms is replaced with the best individual of each of the $(m-1)$th sub-algorithms. The pseudo code of the PPA based on the coarse-grained parallelization model is shown in Algorithm 1.

In our second model, the hybrid parallelization model was used to imple-

Algorithm 1 Pseudo code of the PPA based on the coarse-grained parallelization model

```
Initialize algorithms' parameters
Initialize each of sub-algorithms of solutions;
Evaluate each population in pop_ABC, pop_DE, pop_PSO;
Counter =1;
while Counter ≠ MaximumCycleNumber do
    for each of sub-algorithms do                    ▷ Begin parallel block
        if Counter(mod frequency) = 0 then
            Send the best ind. to the next neighbour (MPI_Send);
            Receive the best ind. from the previous neighbour (MPI_Recv);
            Replace individuals in the population;
        end if
        Run each sub-algorithm with new population;
        Evaluate solutions in the populations;
        Memorize the best solution;
    end for                                          ▷ End parallel block
    Counter = Counter +1
end while
```

ment the two level parallel portfolio algorithm. In this hybrid model, the upper level it is coarse-grained, and the lower level it is master-slave. At the upper level each node runs the different algorithms independently, and at the lower level the master processor sends the offspring to the slave processors and performs selection/updates only. The pseudo code of the PPA based on the hybrid parallelization model is shown in Algorithm 2.

Algorithm 2 Pseudo code of the PPA based on the hybrid parallelization model

```
Initialize algorithms' parameters
Initialize each of sub-algorithms of solutions;
Evaluate the objective function in parallel (Alg. 3.);
Counter =1;
while Counter ≠ MaximumCycleNumber do
    for each of sub-algorithms do                    ▷ Begin parallel block
        if Counter(mod frequency) = 0 then
            Send the best ind. to the next sub-algorithm (MPI_Send);
            Receive the best ind. from the previous sub-algorithm (MPI_Recv);
            Replace individuals in the population;
        end if
        Run each sub-algorithm with new population;
        Evaluate the objective function in parallel (Alg. 3.);
        Memorize the best solution;
    end for                                          ▷ End parallel block
    Counter = Counter +1
end while
```

4. Simulation Results

4.1. The experimental environment

The computer platform used to perform the experiments was based on AMD Opteron 6172 with the number of 12 CPU cores and 128 GB 1333 DDR3 RAM computers with a scientific Linux operating system at TUBITAK ULAKBIM

Algorithm 3 Evaluation of fitness in parallel

```
for all the slaves process do
    Send an individual to a slave process (MPI_Send);
end for
while individual without fitness value do
    Receive result from a slave process (MPI_Recv);
    Send an individual to a slave process (MPI_Send);
end while
for all the slaves process do
    Receive result from a slave process (MPI_Recv);
end for
```

High Performance Computing Center. The proposed algorithm was implemented in C++ using the MPI library.

MPI is a communication protocol for the development of wide variety of parallel computers. MPI's goals are high performance and scalability on all platform. So it provides a platform independent message transfer standards. In this study basic MPI functions were used. The functions MPI_Init and MPI_Finalize are used to initiate and shut down an MPI computation, respectively. The functions MPI_Send and MPI_Recv are used to send and receive a message, respectively. At last, the MPI_Comm_Size is used determine the number of processes in a computation and the MPI_Comm_Rank is used determine the identifier of current process.

4.2. The benchmark suite

The proposed algorithms were tested on twenty CEC2010 special session benchmark functions [52]. All functions were given for the special case of dimension D=1000. The test suite includes separable functions, partially separable functions, partially separable functions that consist of multiple independent subcomponents, each of which is m-nonseparable, and fully nonseparable functions.

4.3. Experimental settings

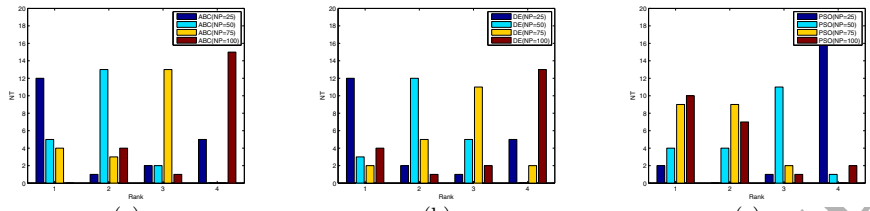
In the experiments, synchronous versions of the basic ABC, DE and PSO algorithms were used. The values of the same parameters used in each algorithm, such as population size and maximum number of evaluations, were chosen to be the same as specified in competition [52]. Population size was 100 and the maximum number of function evaluations was 3.000.000; all the runs were repeated 30 times to compare the algorithms statistically. In ABC, the limit was set to 1000. In DE, strategy was set to DE/rand/1/bin, the weighting factor was set to 0.5 and crossover weight was set to 0.3 as recommended in [23, 24, 25]. In PSO, the learning factors were both set to 1.8 and inertia weight was set to 0.6 as recommended in [26]. Table 1 presents the distribution of the population sizes for all PPA instantiations.

Table 1: Sizes of subpopulations allocated to the constituent algorithms of each instantiation of PPA

Algorithm	Algorithm	ABC	DE	PSO
PPA_{21}	ABC+DE	50	50	-
PPA_{22}	ABC+PSO	70	-	30
PPA_{23}	ABC+PSO	40	-	60
PPA_{24}	DE+PSO	-	70	30
PPA_{25}	DE+PSO	-	40	60
PPA_{31}	ABC+DE+PSO	26	50	24
PPA_{32}	ABC+DE+PSO	30	30	40

4.4. Performance metrics

Various methods and metrics are used to measure the performance of a certain parallel algorithm. Some performance metrics, such as speedup and efficiency are most frequently used to measure the performance of a parallel algorithm. Speedup is the ratio of sequential execution time to parallel execution time (Eq. 6) [53]. The optimal value for the speedup is equal to the number of processors. Efficiency is the ratio of speedup to the number of processors (Eq. 7) and the ideal value for it is 1 [53].



(a) (b) (c)

Figure 2: The performance ranking of serial algorithms with a different number of population a) ABC, b) DE and c) PSO (NT: The number of times the algorithm ranked first)

$$\text{speedup} = \frac{\text{execution time on one processor}}{\text{execution time on m processor}} \tag{6}$$

$$\text{efficiency} = \frac{\text{speedup}}{m} \tag{7}$$

4.5. Effect of the population size to the algorithms

Firstly, we analyzed which size of population is better for the constituent algorithms' serial models on CEC2010 test functions. For this purpose, the population size is investigated for four different cases: 25, 50, 75 and 100. The experimental results are given in Table 2 for the ABC, DE and PSO algorithms. Also, to be able to make good a comparison, the performance ranking of serial algorithms is given in Figure 2.

From Table 2 and Figure 2, it can be concluded that as the population size increases, the PSO algorithm generally produces better results while ABC and DE algorithms generally produces worse results. For the test problems employed in this work, the population size of 75–100 can provide acceptable results for PSO and 25–50 can provide acceptable results for ABC and DE algorithms. This experiment helps to decide the optimum individual number of the population in the PPA.

4.6. Coarse-grained PPA

In the second experiment, we analyzed coarse-grained PPA combinations of two and three constituent algorithms. The statistical indicators (mean and

Table 2: Function error values of the solutions obtained by ABC, DE, PSO for the benchmark functions

Function	Population Size											
	25			50			75			100		
	ABC	DE	PSO	ABC	DE	PSO	ABC	DE	PSO	ABC	DE	PSO
f_1	3.64E-14	6.30E+04	1.21E+10	3.34E-14	2.29E+02	9.19E+09	6.96E-09	1.07E+02	9.47E+09	2.52E-06	3.45E+01	9.35E+09
f_2	7.77E-01	3.56E+00	8.63E+03	8.29E-01	2.34E-01	8.12E+03	2.09E+01	6.75E-02	7.84E+03	7.12E+01	1.27E-02	7.67E+03
f_3	2.09E-12	2.39E-01	1.96E+01	4.40E-10	1.03E-01	1.95E+01	1.28E-05	3.89E-02	1.95E+01	1.45E-03	2.21E-02	1.95E+01
f_4	4.17E+13	5.30E+13	4.51E+12	4.01E+13	6.67E+13	3.46E+12	4.10E+13	7.56E+13	2.57E+12	4.04E+13	8.56E+13	3.02E+12
f_5	6.00E+08	5.13E+08	3.60E+08	5.85E+08	5.02E+08	3.41E+08	5.62E+08	5.21E+08	3.27E+08	5.66E+08	5.37E+08	3.16E+08
f_6	1.99E+07	1.97E+07	1.07E+07	1.99E+07	2.01E+07	6.48E+06	2.00E+07	2.03E+07	4.23E+06	2.01E+07	1.97E+07	4.31E+06
f_7	3.81E+10	4.73E+10	3.67E+10	4.24E+10	5.84E+10	2.61E+10	4.34E+10	6.56E+10	1.95E+10	4.65E+10	6.89E+10	1.49E+10
f_8	7.64E+06	2.29E+07	1.63E+14	5.22E+06	2.06E+07	6.73E+13	5.86E+06	3.23E+07	2.89E+13	1.58E+07	5.74E+07	7.31E+13
f_9	3.60E+08	9.37E+08	6.94E+09	4.84E+08	1.26E+09	6.61E+09	6.02E+08	1.52E+09	6.10E+09	7.17E+08	1.74E+09	5.98E+09
f_{10}	7.24E+03	6.50E+03	8.98E+03	7.12E+03	6.89E+03	8.53E+03	7.17E+03	7.13E+03	8.32E+03	7.27E+03	7.41E+03	8.22E+03
f_{11}	2.00E+02	2.04E+02	2.14E+02	2.01E+02	2.05E+02	2.13E+02	2.01E+02	2.05E+02	2.13E+02	2.02E+02	2.07E+02	2.12E+02
f_{12}	3.64E+05	8.90E+05	1.26E+06	5.23E+05	1.08E+06	1.05E+06	6.39E+05	1.22E+06	8.37E+05	7.38E+05	1.32E+06	9.16E+05
f_{13}	6.40E+02	1.56E+04	1.24E+10	4.72E+02	7.69E+03	1.26E+10	4.65E+02	1.85E+03	8.16E+09	6.32E+02	2.62E+03	8.59E+09
f_{14}	8.05E+08	2.71E+09	1.74E+09	1.16E+09	3.69E+09	1.45E+09	1.49E+09	4.48E+09	1.66E+09	1.82E+09	5.04E+09	1.67E+09
f_{15}	1.47E+04	1.40E+04	9.13E+03	1.46E+04	1.48E+04	8.64E+03	1.45E+04	1.53E+04	8.47E+03	1.46E+04	1.58E+04	8.51E+03
f_{16}	3.99E+02	4.10E+02	3.89E+02	4.01E+02	4.14E+02	3.89E+02	4.03E+02	4.17E+02	3.89E+02	4.05E+02	4.16E+02	3.89E+02
f_{17}	6.93E+05	1.94E+06	1.22E+06	1.03E+06	2.41E+06	1.12E+06	1.26E+06	2.70E+06	1.05E+06	1.46E+06	2.92E+06	1.12E+06
f_{18}	1.04E+04	2.12E+04	4.95E+10	7.74E+03	4.50E+03	4.08E+10	6.55E+03	6.46E+03	3.56E+10	7.11E+03	1.00E+04	3.37E+10
f_{19}	5.72E+06	5.03E+06	1.92E+06	6.88E+06	8.60E+06	2.09E+06	7.80E+06	1.06E+07	2.37E+06	8.54E+06	1.22E+07	2.53E+06
f_{20}	1.22E+01	3.31E+04	5.37E+10	1.49E+01	3.38E+03	4.85E+10	8.50E+01	2.78E+03	4.34E+10	5.33E+02	3.48E+03	4.31E+10

standard deviation values) and average running time for 30 runs of the ABC, DE, PSO and all PPA instantiations are given in Table 4. The performance ranking of the coarse-grained PPA and its two and three constituent algorithms are given in Figures 3 and 4 respectively. Running time comparison between the coarse-grained PPA and its constituent algorithms on the 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}) and 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19}) for all PPA instantiations are given in Figures 5- 8.

The portfolio of the basic models of the ABC, DE and PSO algorithms does not achive the best performance on the CEC2010 functions which take so much time to solve. However, each constituent algorithm delivers the best performance on some problems while it tends not to be successful on others,

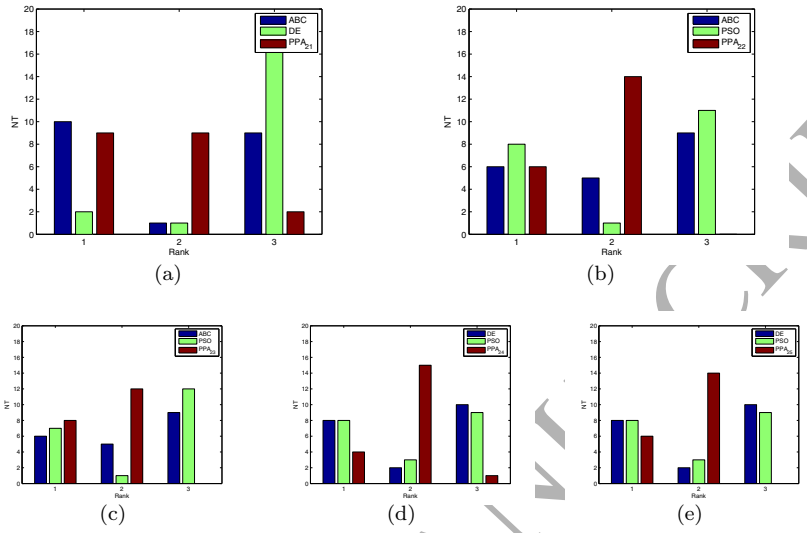


Figure 3: The performance ranking of coarse-grained PPA and its 2 constituent algorithms a) PPA_{21}, b) PPA_{22}, c) PPA_{23}, d) PPA_{24} and e) PPA_{25} (NT: The number of times the algorithm ranked first)

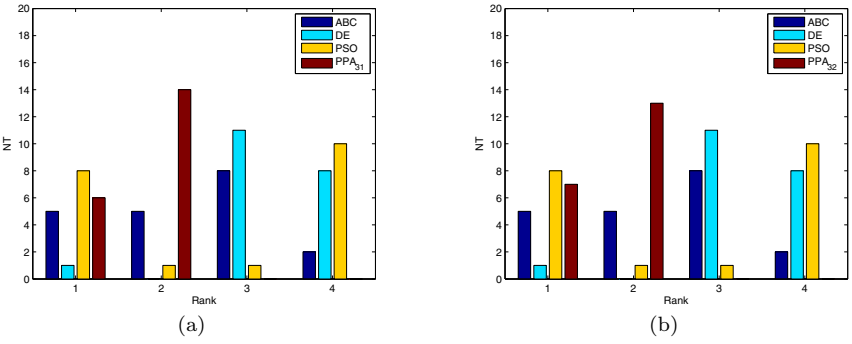


Figure 4: The performance ranking of coarse-grained PPA and its 3 constituent algorithms a) PPA_{31} and b) PPA_{32} (NT: The number of times the algorithm ranked first)

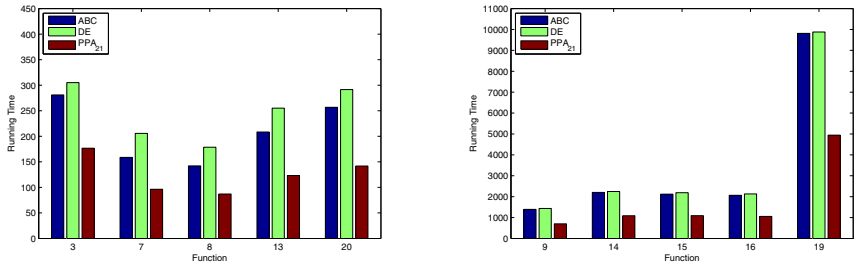


Figure 5: Running time (s) comparison between the coarse-grained PPA_{21} and its 2 constituent algorithms on the 10 test functions a) 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}), b) 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19})

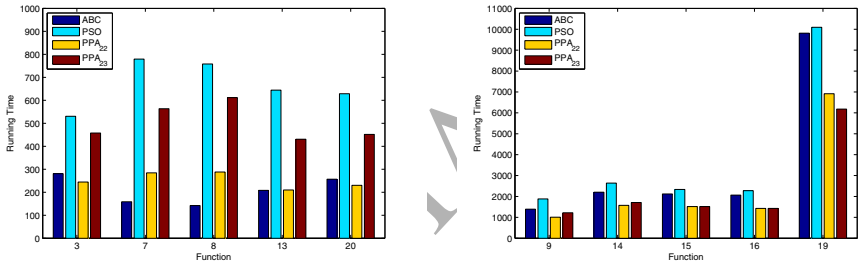


Figure 6: Running time (s) comparison between the coarse-grained PPA_{22}, PPA_{23} and its 2 constituent algorithms on the 10 test functions a) 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}), b) 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19})

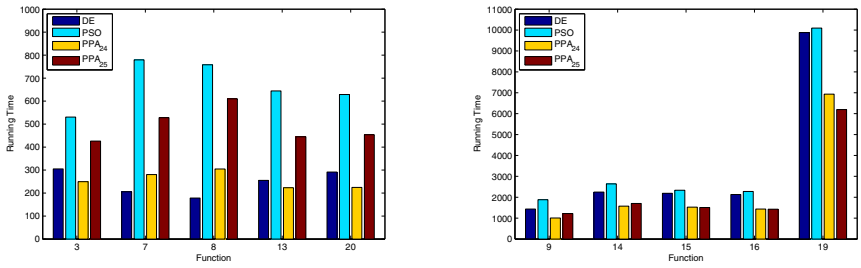


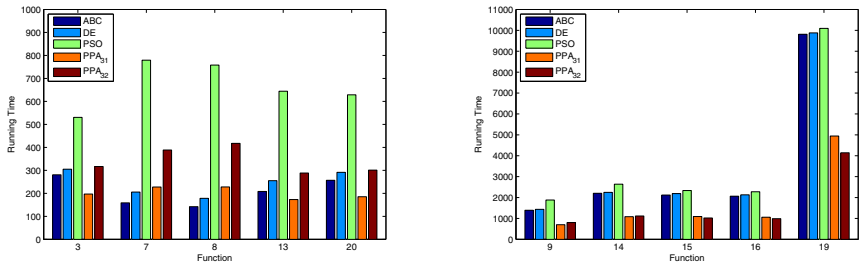
Figure 7: Running time (s) comparison between the coarse-grained PPA_{24}, PPA_{25} and its 2 constituent algorithms on the 10 test functions a) 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}), b) 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19})

Table 3: Running time values obtained by ABC, DE, PSO for the benchmark functions

Function	Population Size											
	25			50			75			100		
	ABC	DE	PSO	ABC	DE	PSO	ABC	DE	PSO	ABC	DE	PSO
f_1	512.71	534.02	1164.26	482.41	518.13	1022.68	486.13	537.05	1002.39	481.53	526.45	993.62
f_2	319.94	301.09	891.06	325.06	339.81	630.72	347.53	374.00	605.79	355.50	378.64	600.57
f_3	223.12	264.95	850.30	235.73	270.44	570.51	265.76	297.38	537.20	280.96	304.99	530.67
f_4	692.27	713.19	1487.05	662.38	705.00	1284.56	668.50	741.51	1200.27	661.57	718.28	1160.67
f_5	532.14	535.89	1110.68	516.31	546.35	856.97	537.12	581.55	828.27	541.72	579.52	817.82
f_6	445.38	455.53	1062.10	432.77	467.30	898.30	455.52	506.42	865.19	467.33	505.61	848.53
f_7	163.91	202.20	968.50	157.94	201.30	869.31	159.95	217.88	816.28	158.64	205.73	779.34
f_8	145.19	183.74	958.73	139.89	182.47	863.69	142.66	183.50	813.20	142.10	178.56	758.06
f_9	1440.27	1431.61	2023.72	1388.95	1422.52	1930.11	1411.86	1429.26	1908.85	1392.27	1436.02	1882.60
f_{10}	1318.60	1304.78	1808.96	1275.28	1310.63	1556.91	1303.40	1318.72	1529.69	1292.82	1337.41	1521.97
f_{11}	1231.24	1236.26	1719.37	1199.81	1245.93	1468.96	1230.58	1249.32	1453.20	1227.34	1277.74	1452.88
f_{12}	397.47	426.79	909.29	383.65	427.16	793.23	392.22	423.19	762.64	384.40	437.88	767.29
f_{13}	211.37	248.04	835.18	207.00	243.59	686.43	212.17	244.36	660.46	208.32	255.08	644.27
f_{14}	2283.64	2232.15	2819.67	2199.11	2245.18	2692.04	2235.92	2242.65	2662.26	2202.63	2249.56	2641.05
f_{15}	2202.55	2155.61	2610.22	2119.90	2169.39	2371.22	2152.96	2186.23	2343.24	2119.82	2190.69	2335.41
f_{16}	2137.39	2097.79	2432.28	2057.04	2106.34	2273.50	2091.12	2117.13	2269.39	2064.19	2129.64	2276.08
f_{17}	657.10	677.95	1211.36	631.86	679.24	1071.23	642.05	680.81	1038.45	633.04	697.33	1036.42
f_{18}	298.97	317.91	815.54	284.15	323.72	660.30	286.29	328.58	654.12	300.52	326.66	648.41
f_{19}	10219.82	9853.06	10234.31	9808.20	9852.01	10125.19	9962.01	9859.02	10123.75	9811.42	9878.49	10093.88
f_{20}	267.68	284.20	815.48	255.50	285.67	653.41	265.12	288.18	630.50	256.80	291.38	628.80

while the portfolio delivers a robust performance. The results in Table 4 and performance ranking figures in Figures 3 and 4 show that, in most of the cases, the portfolio is in rank 1 or 2 while the constituent algorithms ranking changes according to the function considered. These results indicate that all proposed PPA instantiations have a robust performance.

It was seen that each algorithm has different running times and the running time of the PSO algorithm in large scale problems is much more compared to DE and ABC due to its perturbation operator which changes all parameters, while those the others change only some parameters. Distribution also has effect on both the performance and the running time of the portfolio. If more individuals are assigned to the fastest algorithm while less individuals are assigned to the slower ones, a gain is obtained in the running time of the portfolio, as expected.



(a) (b)

Figure 8: Running time (s) comparison between the coarse-grained PPA_{31}, PPA_{32} and its 3 constituent algorithms on the 10 test functions a) 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}), b) 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19})

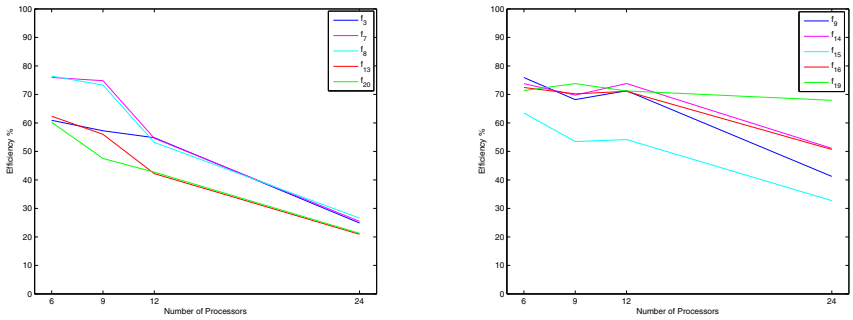
However, assigning more individuals to the fastest algorithm may not always yield the best performance in terms of solution quality.

The speedup in the parallel implementation of the coarse-grained model is expected to be linear since a CPU communicates to only one other CPU and then there is not much communication between CPUs as compared to other models (master-slave, fine-grained and hybrid). The linearity of the running time of a portfolio is measured by the running time of the portfolio versus the constituent algorithms. Because the constituent algorithms have different running times, the linearity of the speedup of the subpopulation varies depending on the control algorithm. If the control algorithm is chosen as the slowest algorithm, the speedup is linear while the speedup is less when one of the other algorithms is chosen as the control algorithm.

4.7. Hybrid PPA

In this section, we analyzed the hybrid PPA with respect to the running times. Because of the proposed model includes three different algorithms and each algorithm needs at least two CPUs such that one is master and one is slave, the minimum number of cores for proposed model can be 6. To test the efficiency of the parallelism, the PPA is executed on 6, 9, 12 and 24 processors. The average running times of the hybrid PPA are given in Table 5. Figure 9

show the graph of the parallel efficiency results based on PSO algorithm on the 5 test functions with the shortest running time (f_3, f_7, f_8, f_{13}, f_{20}) and the 5 test functions with the longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19}) for PPA_{31}.



(a) (b)

Figure 9: Efficiency result for hybrid PPA_{31} algorithm based on PSO algorithm on the 10 test functions a) 5 test functions with shortest running time (f_3, f_7, f_8, f_{13}, f_{20}), b) 5 test functions with longest running time (f_9, f_{14}, f_{15}, f_{16}, f_{19})

Because the hybrid model is a composition of both the coarse-grained and master-slave models, the performance depends on all the factors by which each model is affected, such as the running time of constituent algorithms, distribution of the population and the number of CPUs. The average running times in Table 5 and parallel efficiency graph in Figure 9 show that the efficiency in this model is less for functions with low computation times (f_3, f_7, f_8, f_{13}, f_{20}) and much more for functions with high computation times (f_9, f_{14}, f_{15}, f_{16}, f_{19}). The average parallelization efficiency is about 70% for the f_{19} function, which has the highest computation cost. Also, increasing the number of CPUs dramatically provides no more gain and even causes the parallelization efficiency to decrease, especially in cases with shortest running times. These results clearly show that the function evaluation time and communication between the processors play an important role. Using more processors for a small problem increases the running time as a result of excessive communication between the processors. Thus we can say that the hybrid PPA algorithm can achieve a

reasonable speedup for expensive fitness functions with a moderate number of processors.

4.8. PPA for Time Series Predection Problems

In the last experiment, the performance of the PPA algorithm was applied on two well-known time series prediction problems: Mackey Glass (MG) time series and Box-Jenkins (BJ) time series which have been in literature to compare the performance of different neural networks and neurofuzzy systems.

The MG time series is a sequence produced by a nonlinear time delay differential equation [54]. The task of this benchmark is to predict the value of the time serie at the point $(t+6)$ from the earlier points $(t-18)$, $(t-12)$, $(t-6)$ and (t). The BJ time series was recorded from a combustion process of methane air mixture [55]. The input $u(t)$ is the gas flowing rate and the output $y(t)$ is CO_2 concentration. In most of the studies, it has been stated that the best set of input variables for predicting $y(t)$ is $y(t-1)$ and $u(t-4)$, which is used in our study.

In this study, log-sigmoid transfer function, Mean Squared Error (MSE) performance function and seven network structures that include one hidden layer MLP networks with 2,4,6,8,12,16 and 20 hidden nodes were used. The maximum generation was set to 1000 and the other parameters of PPA were set to default values in experimental settings. The data were normalized to the range [0, 1] and shuffled randomly before training of the network. All the experiments are implemented for 30 runs. The neural networks were performed using 450 training data, 500 testing data points for MG time series and 140 training data, 150 testing data points for BJ time series. In Table 6 average MSE values of the neural networks with different structures trained using the PPA algorithm are presented. The performance of the PPA were compared with back propagation (BP), PSO, cooperative random learning particle swarm optimization (CRPSO), and genetic algorithm (GA) results data is from recently reported publication [56] in Table 7.

Table 6 shows that on the prediction problems, the MPL that showed the

best performance is the topology using 12 hidden nodes for MG and 8 hidden nodes for BJ time series. From Table 7, it is observed that in both training and testing cases PPA performed better than the other algorithms used in our study. So, it can be concluded that PPA is effective for time series problems.

5. Conclusions

We have proposed a method for enhancing the robustness of optimization algorithms by running more algorithms in parallel to solve the same optimization problems. For this purpose, we constructed the PPA with basic models of the ABC, DE and PSO algorithms. The PPA does not guarantee the best results. However, each constituent algorithm delivers the best performance on some problems while it tends not to be successful on others, while the PPA delivers a robust performance. The results clearly showed that the PPA has advantages when a number of different problems need to be solved. Also, the performance of the PPA is quite sensitive to the distribution of the constituent algorithms. However, description of any one portfolio which is the best, is very difficult. In particular, if you use more constituent algorithms in a PPA, finding the optimal PPA is not possible.

We explored two types of parallel implementation model: coarse-grained and a hybrid model. These models are very suitable for portfolio structures. Empirical studies demonstrated that each implementation of the PPA gained speedup when compared to the sequential counterparts. In addition, we explored how to affect the constituent algorithms' distribution in the population in terms of running time. The result clearly showed that, distribution not only affects the PPAs performance but also affects the running time significantly. We can clearly say that when the number of the slow algorithms' individuals increases in the population, the running time of the PPA increases. Similarly, if the number of the slow algorithms' individuals decreases, the running time of the PPA decreases because communication between algorithms requires synchronization.

We also investigate the performance of the PPA for training seven different

architectures of neural networks in a two time series prediction problems. This study represents that portfolio algorithms are able to produce good prediction accuracy.

In the future we would like to study different optimization algorithms which complement each other significantly and apply the proposed parallel models to different real-world problems.

6. Acknowledgement

The numerical calculations reported in this paper were performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TR-Grid e-Infrastructure). This study is a part of the project numbered FBA–11–3520. The author would like to thank to Erciyes University Scientific Research Projects Unit.

References

References

[1] C. Gomes, B. Selman, Algorithm portfolio design: Theory vs. practice, in: In UAI, 1997, pp. 190–197.

[2] B. A. Huberman, R. M. Lukose, T. Hogg, An economics approach to hard computational problems, Science 275 (1997) 51–54.

[3] A. S. Fukunaga, Genetic algorithm portfolios, in: IEEE Congr. Evol. Comput, La Jolla, CA, 2000, pp. 16–19.

[4] C. Gomes, B. Selman, Algorithm portolios, Artificial Intelligence 126 (1–2) (2001) 43–62.

[5] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: 11th European

PVM/MPI Users' Group Meeting Proceedings, Budapest, Hungary, 2004, pp. 97–104.

[6] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. Rep. TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005).

[7] K. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer–Verlag, Berlin, Germany, 2005.

[8] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, WA, Australia, 1995, pp. 1942–1948.

[9] R. Poli, An analysis of publications on particle swarm optimisation applications, Technical report csm-469, Department of Computer Science, University of Essex, UK (2007).

[10] D. Karaboga, B. Akay, A survey: Algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (1) (2009) 68–55.

[11] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 4–31.

[12] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: Artificial bee colony (abc) algorithm and applications, Artificial Intelligence Review 42 (2014) 21–57.

[13] I. Boussaid, J. Lepagnot, P. Siarry, A survey on optimization metaheuristics, Information Sciences 237 (2013) 82–117.

[14] F. Peng, K. Tang, G. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Transactions on Evolutionary Computation 14 (5) (2010) 782–800.

[15] J. A. Vrugt, B. A. Robinson, J. M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 243–259.

[16] S. Y. Yuen, C. K. Chow, X. Zhang, Which algorithm should i choose at any point of the search: an evolutionary portfolio approach, ... conference on Genetic and evolutionary ... (2013) 567–574.

[17] M. Muñoz, M. Kirley, S. K. Halgamuge, The Algorithm Selection Problem on the Continuous Optimization Domain, Vol. 445, 2013.

[18] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.

[19] M. Z. Ali, N. H. Awad, A novel class of niche hybrid cultural algorithms for continuous engineering optimization, Information Sciences 267 (2014) 158–190.

[20] B. Lacroix, D. Molina, F. Herrera, Region based memetic algorithm for real parameter optimization, Information Sciences 262 (2014) 15–31.

[21] H. Samulowitz, C. Reddy, A. Sabharwal, M. Sellmann, Snappy: a simple algorithm portfolio., in: Theory and Applications of Satisfiability Testing, Vol. 7962, 2013, pp. 422–428.

[22] K. Tang, F. Peng, G. Chen, X. Yao, Population-based algorithm portfolios with automated constituent algorithms selection, Information Sciences 279 (2014) 94–104.

[23] R. Storn, On the usage of differential evolution for function optimization, Proceedings of North American Fuzzy Information Processing NAFIPS'96 (1996) 519–523.

[24] R. Storn, K. Price, Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[25] R. Gämperle, S. D. Müller, P. Koumoutsakos, A Parameter Study for Differential Evolution, in: WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, 2002, pp. 293–298.

[26] J. Vesterstrom, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, Evolutionary Computation, 2004. CEC2004. Congress on 2 (2004) 1980–1987.

[27] B. Akay, D. Karaboga, Parameter Tuning for the Artificial Bee Colony Algorithm, 1st International Conference on Computational Collective Intelligence Semantic Web Social Networks Multiagent Systems 5796 (Figure 1) (2009) 608–619.

[28] D. Thierens, Adaptive strategies for operator allocation, Studies in Computational Intelligence 54 (2007) 77–90.

[29] A. Fialho, L. Costa, M. Schoenauer, M. Sebag, Analyzing bandit-based adaptive operator selection mechanisms, Annals of Mathematics and Artificial Intelligence 60 (1) (2010) 25–64.

[30] D. Opitz, R. Maclin, Popular Ensemble Methods: An Empirical Study, Journal of Artificial Intelligence Research 11 (1999) 169–198.

[31] L. Rokach, Ensemble-based classifiers, Artificial Intelligence Review 33 (1-2) (2010) 1–39.

[32] E. L. Yu, P. N. Suganthan, Ensemble of niching algorithms, Information Sciences 180 (2010) 2815–2833.

[33] R. Mallipeddi, P. N. Suganthan, Ensemble of constraint handling techniques, IEEE Transactions on Evolutionary Computation 14 (4) (2010) 561–579.

[34] A. P. Piotrowski, Adaptive memetic differential evolution with global and local neighborhood-based mutation operators, Information Sciences 241 (2013) 164–194.

[35] S. Zhou, Z. Sun, Using ensemble method to improve the performance of genetic algorithm, Computational Intelligence and Security 3801 (2015) 255–260.

[36] G. Wu, R. Mallipeddi, P. N. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Information Sciences 329 (2016) 329–345.

[37] S. Z. Zhao, P. N. Suganthan, Q. Zhang, Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes, IEEE Transactions on Evolutionary Computation 16 (3) (2012) 442–446.

[38] R. Shang, Y. Wang, J. Wang, L. Jiao, S. Wang, L. Qi, A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem, Information Sciences 277 (2014) 609–642.

[39] M. Petrik, S. Zilberstein, Learning parallel portfolios of algorithms, Annals of Mathematics and Artificial Intelligence 48 (1–2) (2006) 85–106.

[40] X. Yun, S. L. Epstein, Learning Algorithm Portfolios for Parallel Execution, Learning and Intelligent Optimization (2012) 323–338.

[41] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, The Design and Analysis of an Algorithm Portfolio for SAT, in: Principles and Practice of Constraint Programming, 2007, pp. 712–727.

[42] L. Shi, L. Xi, X. Ma, M. Weng, X. Hu, A novel ensemble algorithm for biomedical classification based on Ant Colony Optimization, Applied Soft Computing Journal 11 (8) (2011) 5674–5683.

[43] M. Graff, H. Escalante, J. Cerda-Jacobo, A. Gonzalez, Models of performance of time series forecasters, Neurocomputing 122 (2013) 375–385.

[44] T. Chen, H. Xue, Z. Hong, M. Cui, H. Zhao, A hybrid ensemble method based on double disturbance for classifying microarray data, Bio-Medical Materials and Engineering 26 (2015) 1961–1968.

[45] J. Tang, G. Zhang, B. Lin, B. Zhang, A hybrid algorithm for flexible job-shop scheduling problem, in: Procedia Engineering, Vol. 15, 2011, pp. 3678–3683.

[46] J. Castro, O. Castillo, P. Melin, A. Rodrguez-Daz, A hybrid learning algorithm for a class of interval type-2 fuzzy neural networks, Information Sciences 179 (13) (2009) 2175–2193.

[47] J. Zhao, X. Yu, Adaptive natural gradient learning algorithms for mackey-glass chaotic time prediction, Neurocomputing 157 (2015) 41–45.

[48] W. Zhang, H. Ji, G. Liao, Y. Zhang, A novel extreme learning machine using privileged information, Neurocomputing 168 (2015) 823–828.

[49] R. Adhikari, A neural network based linear ensemble framework for time series forecasting, Neurocomputing 157 (2015) 231–242.

[50] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm, Journal of Global Optimization 39 (2007) 459–471.

[51] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (2002) 443–462.

[52] K. Tang, X. Li, P. N. Suganthan, T. Weise, Benchmark functions for the cec'2010 special session and competition on large scale global optimization, Tech. rep., Nature Inspired Computation and Applications Laboratory, USTC, China, http://nical.ustc.edu.cn/cec10ss.php (2009).

[53] E. Alba, Parallel Metaheuristics: A New Class of Algorithms, Wiley-Interscience, 2005.

[54] M. C. Mackey, L. Glass, Oscillation and chaos in physiological control systems, Science 197 (1977) 287–289.

[55] G. Box, G. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, San Francisco, 1970.

[56] L. Zhao, Y. Yang, Pso-based single multiplicative neuron model for time series prediction, Expert Systems with Applications 36 (2) (2009) 2805–2812.

Table 4: Function error values of the solutions obtained by ABC, DE, PSO and seven PPA instantiations for the benchmark functions

Function	Algorithms									
	ABC	DE	PSO	PPA_{21}	PPA_{22}	PPA_{23}	PPA_{24}	PPA_{25}	PPA_{31}	PPA_{32}
f_1	2.52E-06	3.45E+01	9.35E+09	3.42E-05	4.45E-06	1.46E-04	2.18E+08	2.33E+08	1.52E-03	1.43E-03
f_2	7.12E+01	1.27E-02	7.67E+03	8.90E+01	8.56E+01	9.39E+01	3.43E+03	3.47E+03	9.01E-02	4.13E-01
f_3	1.45E-03	2.21E-02	1.95E+01	4.62E-03	2.50E-03	4.13E-03	1.61E+01	1.63E+01	7.01E-04	3.22E-04
f_4	4.04E+13	8.56E+13	3.02E+12	3.50E+13	3.76E+12	3.09E+12	5.17E+12	3.23E+12	3.60E+12	3.85E+12
f_5	5.66E+08	5.37E+08	3.16E+08	5.33E+08	3.42E+08	3.23E+08	3.52E+08	3.25E+08	3.66E+08	3.41E+08
f_6	2.01E+07	1.97E+07	4.31E+06	2.00E+07	9.54E+06	4.57E+06	7.21E+06	5.40E+06	6.27E+06	5.64E+06
f_7	4.65E+10	6.89E+10	1.49E+10	3.20E+10	6.23E+09	4.09E+09	1.06E+10	1.20E+10	5.15E+09	6.66E+09
f_8	1.58E+07	5.74E+07	7.31E+13	2.65E+07	9.46E+06	9.36E+06	4.50E+11	3.47E+11	1.73E+07	1.75E+07
f_9	7.17E+08	1.74E+09	5.98E+09	6.83E+08	6.61E+08	6.60E+08	9.89E+08	8.69E+08	6.91E+08	6.57E+08
f_{10}	7.27E+03	7.41E+03	8.22E+03	7.28E+03	4.65E+03	4.57E+03	7.10E+03	6.88E+03	4.69E+03	4.57E+03
f_{11}	2.02E+02	2.07E+02	2.12E+02	2.03E+02	1.96E+02	1.95E+02	2.13E+02	2.12E+02	1.97E+02	1.94E+02
f_{12}	7.38E+05	1.32E+06	9.16E+05	7.11E+05	4.97E+05	3.98E+05	3.97E+05	3.64E+05	4.66E+05	3.81E+05
f_{13}	6.32E+02	2.62E+03	8.59E+09	1.00E+03	7.89E+02	1.23E+03	3.24E+07	3.08E+07	1.72E+03	1.51E+03
f_{14}	1.82E+09	5.04E+09	1.67E+09	1.77E+09	1.78E+09	1.71E+09	2.05E+09	1.69E+09	1.68E+09	1.68E+09
f_{15}	1.46E+04	1.58E+04	8.51E+03	1.45E+04	8.73E+03	8.52E+03	8.69E+03	8.53E+03	8.84E+03	8.64E+03
f_{16}	4.05E+02	4.16E+02	3.89E+02	4.05E+02	3.90E+02	3.89E+02	3.90E+02	3.89E+02	3.91E+02	3.89E+02
f_{17}	1.46E+06	2.92E+06	1.12E+06	1.48E+06	1.23E+06	1.10E+06	1.22E+06	1.12E+06	1.27E+06	1.16E+06
f_{18}	7.11E+03	1.00E+04	3.37E+10	1.19E+04	9.55E+03	1.54E+04	2.03E+08	1.92E+08	9.24E+03	9.62E+03
f_{19}	8.54E+06	1.22E+07	2.53E+06	7.32E+06	2.92E+06	2.66E+06	2.85E+06	2.66E+06	3.00E+06	2.83E+06
f_{20}	5.33E+02	3.48E+03	4.31E+10	1.15E+03	8.55E+02	1.54E+03	2.67E+08	2.26E+08	1.89E+03	1.85E+03

Table 5: Average running times (s) of serial PSO and PPA_{31} algorithms for hybrid model

Alg.	CPU(s)	Functions									
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
PSO	1	993.62	600.57	530.67	1160.67	817.82	848.53	779.34	758.06	1882.60	1521.97
PPA_{31}	6	226.31	164.55	145.21	251.06	265.34	177.37	171.00	165.26	413.31	421.48
	9	157.60	141.31	102.99	208.13	221.24	144.18	115.74	115.00	306.79	355.04
	12	140.97	122.79	80.68	168.37	202.52	130.99	118.92	118.79	220.08	274.26
	24	135.27	125.93	88.94	148.52	183.87	121.99	127.61	118.77	190.23	229.08
Alg.	CPU(s)	Functions									
		f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
PSO	1	1452.88	767.29	644.27	2641.05	2335.41	2276.08	1036.42	648.41	10093.88	628.80
PPA_{31}	6	313.77	188.24	172.22	596.62	613.06	523.78	284.64	204.70	2357.72	173.97
	9	266.23	171.32	127.80	420.74	485.54	360.24	230.04	141.36	1520.00	147.04
	12	182.86	134.08	127.26	298.20	359.44	266.86	180.34	132.60	1180.98	122.66
	24	170.22	128.08	128.34	215.46	297.28	187.10	158.54	127.00	619.10	123.34

Table 6: The training and testing MSE values obtained by PPA_{31} for the time series

Number of Hidden Nodes	Mackey Glass		Box Jenkins	
	Train	Test	Train	Test
2	3.4945E-03	3.5128E-03	8.9114E-04	9.1565E-04
4	1.3430E-03	1.4180E-03	6.6052E-04	7.9206E-04
6	6.9713E-04	7.7762E-04	6.1556E-04	7.9727E-04
8	5.7153E-04	6.2639E-04	6.0978E-04	7.0342E-04
12	5.4804E-04	6.1113E-04	5.9920E-04	7.2255E-04
16	6.0202E-04	7.0184E-04	5.7387E-04	7.3291E-04
20	7.5745E-04	8.9585E-04	5.8302E-04	9.0047E-04

Table 7: The comparison results of MSE values for the time series

Time Series	Method	BP	PSO	CRPSO	GA	PPA_{31}
Mackey Glass	Train	0.0038	0.0017	5.2504E-04	5.9584E-04	5.4804E-04
	Test	0.0041	0.0018	5.4910E-04	6.2173E-04	6.1113E-04
Box Jenkins	Train	0.0030	0.0029	0.0017	0.0018	6.0978E-04
	Test	0.0056	0.0054	0.0021	0.0023	7.0342E-04

Rustu Akay received the M.S. and the Ph.D. degree in computer engineering from Erciyes University, Kayseri, Turkey, in 2006 and 2014, respectively. Since 2014 he is an Assistant Professor in the Department of Mechatronics Engineering, Erciyes University. His current research interests include evolutionary algorithm, parallel programming, intelligent control systems and their applications.

Alper Basturk received his BS degree in electronics engineering from Erciyes University, Kayseri, Turkey, in July-1998. He then joined as a research assistant to the Dept. of Electronics Eng. of Erciyes University. He received his MS and PhD degrees in electronics engineering from Erciyes University in August-2001 and November-2006. He is currently working in the Computer Engineering department. His research areas are digital signal and image processing, neural networks, fuzzy and neuro-fuzzy systems, intelligent optimization and applications of these techniques.

Adem Kalinli is a Professor of Computer Engineering at the Erciyes University, Turkey. He has authored/co-authored more than 50 refereed publications. His major research interests include biometrics, image processing, pattern recognition, neural networks and their real-world applications. Since 2009, Prof. Kalinli is coordinator of Erciyes University Scientific Research Projects Unit.

Xin Yao (F'03) is a Chair (Professor) of Computer Science and the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications) at the University of Birmingham, UK. He is an IEEE Fellow and the President (2014-15) of IEEE Computational Intelligence Society (CIS). His major research interests include evolutionary computation and ensemble learning. He published 200+ refereed international journal papers. His papers won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards. He received the prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE CIS Evolutionary Computation Pioneer Award in 2013. He was the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. He has published frequently on the topic of capacitated arc routing problems (CARP) since his first paper in 2006 (H. Handa, L. Chapman and Xin Yao, "Robust route optimisation for gritting/salting trucks: A CERCIA experience," IEEE Computational Intelligence Magazine, 1(1):6-9, February 2006.).

Rustu Akay

Alper Basturk

Adem Kalinli

Xin Yao